What Is Claimed Is:

1. A method for reducing the overhead involved in executing native code methods in an application running on a virtual machine, comprising:

selecting a call to a native code method to be optimized within the virtual machine;

decompiling at least part of the native code method into an intermediate representation;

obtaining an intermediate representation associated with the application running on the virtual machine which interacts with the native code method;

combining the intermediate representation for the native code method with the intermediate representation associated with the application running on the virtual machine to form a combined intermediate representation; and

generating native code from the combined intermediate representation, wherein the native code generation process optimizes interactions between the application running on the virtual machine and the native code method.

2. The method of claim 1, wherein selecting the call to the native code method involves selecting the call based upon at least one of:

the execution frequency of the call; and

the overhead involved in performing the call to the native code method as compared against the amount of work performed by the native code method.

3. The method of claim 1, wherein optimizing interactions between the application running on the virtual machine and the native code method involves optimizing calls to the native code method by the application.

4. The method of claim 1, wherein optimizing interactions between the application running on the virtual machine and the native code method involves optimizing callbacks by the native code method into the virtual machine.

5. The method of claim 4, wherein optimizing callbacks by the native code method into the virtual machine involves optimizing callbacks that access heap objects within the virtual machine.

6. The method of claim 4,

wherein the virtual machine is a Java Virtual Machine (JVM); and

wherein combining the intermediate representation for the native code method with the intermediate representation associated with the application running on the virtual machine involves integrating calls provided by the Java Native Interface (JNI) into the native code method.

7. The method of claim 1, wherein obtaining the intermediate representation associated with the application running on the virtual machine involves recompiling a corresponding portion of the application.

8. The method of claim 1, wherein obtaining the intermediate representation associated the application running on the virtual machine involves accessing a previously generated intermediate representation associated with the application running on the virtual machine.

9. The method of claim 1, wherein prior to decompiling the native code method, the method further comprises setting up a context for the decompilation by:

determining a signature of the call to the native code method; and

determining a mapping from arguments of the call to corresponding locations in a native application binary interface (ABI).

10. A computer-readable storage medium storing instructions that when executed by a computer cause the computer to perform a method for reducing the overhead involved in executing native code methods in an application running on a virtual machine, the method comprising:

selecting a call to a native code method to be optimized within the virtual machine;

decompiling at least part of the native code method into an intermediate representation;

obtaining an intermediate representation associated with the application running on the virtual machine which interacts with the native code method;

combining the intermediate representation for the native code method with the intermediate representation associated with the application running on the virtual machine to form a combined intermediate representation; and

generating native code from the combined intermediate representation, wherein the native code generation process optimizes interactions between the application running on the virtual machine and the native code method.

11. The computer-readable storage medium of claim 10, wherein selecting the call to the native code method involves selecting the call based upon at least one of:

the execution frequency of the call; and

the overhead involved in performing the call to the native code method as compared against the amount of work performed by the native code method.

12. The computer-readable storage medium of claim 10, wherein optimizing interactions between the application running on the virtual machine and the native code method involves optimizing calls to the native code method by the application.

13. The computer-readable storage medium of claim 10, wherein optimizing interactions between the application running on the virtual machine and the native code method involves optimizing callbacks by the native code method into the virtual machine.

14. The computer-readable storage medium of claim 13, wherein optimizing callbacks by the native code method into the virtual machine involves optimizing callbacks that access heap objects within the virtual machine.

15. The computer-readable storage medium of claim 13,

wherein the virtual machine is a Java Virtual Machine (JVM); and

wherein combining the intermediate representation for the native code method with the intermediate representation associated with the application running on the virtual machine involves integrating calls provided by the Java Native Interface (JNI) into the native code method.

16. The computer-readable storage medium of claim 10, wherein obtaining the intermediate representation associated with the application running on the virtual machine involves recompiling a corresponding portion of the application.

17. The computer-readable storage medium of claim 10, wherein obtaining the intermediate representation associated the application running on the virtual machine involves accessing a previously generated intermediate representation associated with the application running on the virtual machine.

18. The computer-readable storage medium of claim 10, wherein prior to decompiling the native code method, the method further comprises setting up a context for the decompilation by:

determining a signature of the call to the native code method; and

determining a mapping from arguments of the call to corresponding locations in a native application binary interface (ABI).

19. An apparatus that reduces the overhead involved in executing native code methods in an application running on a virtual machine, comprising:

a selection mechanism configured to select a call to a native code method to be optimized within the virtual machine;

a decompilation mechanism configured to decompile at least part of the native code method into an intermediate representation;

an obtaining mechanism configured to obtain an intermediate representation associated with the application running on the virtual machine which interacts with the native code method;

an combining mechanism configured to combine the intermediate representation for the native code method with the intermediate representation associated with the application running on the virtual machine to form a combined intermediate representation; and

a code generation mechanism configured to generate native code from the combined intermediate representation, wherein the code generation mechanism

optimizes interactions between the application running on the virtual machine and the native code method.

20. The apparatus of claim 19, wherein the selection mechanism is configured to select the call to the native code method based upon at least one of:

the execution frequency of the call; and

the overhead involved in performing the call to the native code method as compared against the amount of work performed by the native code method.

21. The apparatus of claim 19, wherein the optimization mechanism is configured to optimize calls to the native code method by the application.

22. The apparatus of claim 19, wherein the optimization mechanism is configured to optimize callbacks by the native code method into the virtual machine.

23. The apparatus of claim 22, wherein the optimization mechanism is configured to optimize callbacks that access heap objects within the virtual machine.

24. The apparatus of claim 22,

wherein the virtual machine is a Java Virtual Machine (JVM); and

wherein the combining mechanism is configured to integrate calls provided by the Java Native Interface (JNI) into the native code method.

25. The apparatus of claim 19, wherein the obtaining mechanism is configured to obtain the intermediate representation associated with the

application running on the virtual machine by recompiling a corresponding portion of the application.

26. The apparatus of claim 19, wherein the obtaining mechanism is configured to obtain the intermediate representation associated the application running on the virtual machine by accessing a previously generated intermediate representation associated with the application running on the virtual machine.

27. The apparatus of claim 19, further comprising a context setting mechanism, wherein prior to decompilation of the native code method, the context setting mechanism is configured to:

determine a signature of the call to the native code method; and

determine a mapping from arguments of the call to corresponding locations in a native application binary interface (ABI).

28. A method for reducing the overhead involved in executing native code methods in an application running on a virtual machine, comprising:

deciding to optimize a callback by a native code method into the virtual machine;

decompiling at least part of the native code method into an intermediate representation;

obtaining an intermediate representation associated with the application running on the virtual machine which interacts with the native code method;

combining the intermediate representation for the native code method with the intermediate representation associated with the application running on the virtual machine to form a combined intermediate representation; and

generating native code from the combined intermediate representation, wherein the native code generation process optimizes the callback by the native code method into the virtual machine.

29. The method of claim 28, wherein the native code generation process also optimizes calls to the native code method by the application.

30. The method of claim 28, wherein optimizing the callback by the native code method into the virtual machine involves optimizing a callback that accesses a heap object within the virtual machine.

31. The method of claim 28,

wherein the virtual machine is a Java Virtual Machine (JVM); and

wherein combining the intermediate representation for the native code method with the intermediate representation associated with the application running on the virtual machine involves integrating calls provided by the Java Native Interface (JNI) into the native code method.

32. A computer-readable storage medium storing instructions that when executed by a computer cause the computer to perform a method for reducing the overhead involved in executing native code methods in an application running on a virtual machine, the method comprising:

deciding to optimize a callback by a native code method into the virtual machine;

decompiling at least part of the native code method into an intermediate representation;

obtaining an intermediate representation associated with the application running on the virtual machine which interacts with the native code method;

combining the intermediate representation for the native code method with the intermediate representation associated with the application running on the virtual machine to form a combined intermediate representation; and

generating native code from the combined intermediate representation, wherein the native code generation process optimizes the callback by the native code method into the virtual machine.

33. The computer-readable storage medium of claim 32, wherein the native code generation process also optimizes calls to the native code method by the application.

34. The computer-readable storage medium of claim 32, wherein optimizing the callback by the native code method into the virtual machine involves optimizing a callback that accesses a heap object within the virtual machine.

35. The computer-readable storage medium of claim 32,

wherein the virtual machine is a Java Virtual Machine (JVM); and

wherein combining the intermediate representation for the native code method with the intermediate representation associated with the application running on the virtual machine involves integrating calls provided by the Java Native Interface (JNI) into the native code method.

36. An apparatus that reduces the overhead involved in executing native code methods in an application running on a virtual machine, comprising:

a selection mechanism configured to decide to optimize a callback by a native code method into the virtual machine;

a decompilation mechanism configured to decompile at least part of the native code method into an intermediate representation;

an obtaining mechanism configured to obtain an intermediate representation associated with the application running on the virtual machine which interacts with the native code method;

an combining mechanism configured to combine the intermediate representation for the native code method with the intermediate representation associated with the application running on the virtual machine to form a combined intermediate representation; and

a code generator configured to generate native code from the combined intermediate representation, wherein the code generator optimizes the callback by the native code method into the virtual machine.

37. The apparatus of claim 36, wherein the code generator also optimizes calls to the native code method by the application.

38. The apparatus of claim 36, wherein while optimizing the callback by the native code method into the virtual machine, the code generator optimizes a callback that accesses a heap object within the virtual machine.

39. The apparatus of claim 36,

wherein the virtual machine is a Java Virtual Machine (JVM); and

wherein the combining mechanism is configured to integrate calls provided by the Java Native Interface (JNI) into the native code method.